2004 Command and Control Research and Technology Symposium
The Power of Information Age Concepts and Technologies

APPLICATION QOS BASED TIME-CRITICAL MACHINE-TO-MACHINE RESOURCE MANAGEMENT IN BM/C2 SYSTEMS

E. Douglas Jensen
The MITRE Corporation
202 Burlington Road, Bedford, MA 01730
Voice 781-271-2514, Fax 781-271-4686
jensen@[mitre,real-time].org
http://www.real-time.org

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
JUN 2004		**00-00-2004 to 00-00-2004**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Application QOS Based Time-Critical Machine-to-Machine Resource Management in BM/C2 Systems	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **The MITRE Corporation,202 Burlington Road,Bedford,MA,01730**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**42**	

Application QoS Based Time-Critical Machine-to-Machine Resource Management in BM/C^2 Systems

E. Douglas Jensen
The MITRE Corporation
Bedford, MA 01730
jensen@[mitre,real-time].org

Abstract

This paper summarizes a novel paradigm for expressing, enforcing, and formally reasoning about time-criticality of machine-to-machine resource management in battle management (BM) and C^2 systems. Such systems are largely dynamic and asynchronous, and have time frames of $O(10^{-1})$ seconds and higher. Thus they fall into a neglected gap between traditional static periodic "real-time" systems, and traditional "any time" scheduling/planning systems (e.g., for logistics). The paradigm uses application-level QoS metrics (such as track quality, circular error probable, etc.) to derive utility functions for completing tasks, and then uses those utility functions for resource management. This paradigm has been successfully employed in several experimental BM/C^2 demonstration systems, and is the topic of on-going research by industry and academia.

Keywords

Time-critical, real-time, resource management, scheduling, QoS, utility, time/utility functions, utility accrual

Introduction

A system provides services, each of which may have various qualities. Each service usually has multiple dimensions of quality. Generic examples include timeliness, availability, and security. Quality of service (QoS) is traditionally associated only with network communications – e.g., bandwidth, BER, latency and jitter. But QoS is an inherently general concept that can be employed at any levels of a system and enterprise.

In many systems, complex dynamic service – and thus resource – conflicts and dependencies arise. Resolution of the resource contention affects the quality of the provided services to the users and thus the success of the mission and the enterprise. Not all service requests can always be perfectly satisfied; adaptive situation and application-specific resource management decisions are required.

Some of the service requests are time-critical – the quality of the service depends on when it is performed. The timeliness of these services may have potentially drastic impacts on the enterprise and its mission, including survival of property and human lives.

By popular convention, the term "safety critical" is reserved for very simple, static, systems that can be developed in accordance with DO-178B and other similar standards. But many large scale, dynamic, systems (such as for combat platform mission management, and battle management) have at least as much potential for human harm as do those "safety critical" ones, yet they are beyond the foreseeable state of the art in high assurance in the spirit of DO-178B et al. In the absence of substantive research results on that problem, assurance must be sought by design methodologies, simulations, and extensive testing.

In this paper we discuss resolving resource contention by adaptively maximizing total *application-level quality of service* ("AQoS") of a set of services according to application- and situation-specific criteria. Multimedia and similar applications popular in the real-time research community have AQoS metrics, but they are isomorphic to the QoS metrics of traditional networking, and so do not present significantly new resource management challenges and op-

portunities. We are concerned with managing application, as well as system, resources (hardware and software) – above, as well as at, the network level – using AQoS metrics, such as (in most systems of interest to us) track quality and weapon spherical error probable.

Application designers often think in terms of what we refer to as AQoS metrics, but not in a general and methodological way. Instead, they consider certain metrics and use their domain expertise to attempt to aggregate these into the proper "tuning" of the system. System and resource management (OS, middleware) software designers usually do not think in terms of AQoS. Thus, there are needs for systematizing and quantifying: the expression of AQoS metrics, and the potential trade-offs among them, to resource management software; and managing resources adaptively by employing AQoS to dynamically optimize the system's behavior according to the users' wishes.

We are particularly interested in helping meet those needs in the context of satisfying application timeliness requirements. We are focused on using AQoS metrics in defining how each service's timeliness contributes to its utility to the current state of the mission and sequencing access to shared resources based in part on maximizing the utility accrued from the set of services.

Dynamic Time-Critical Control Systems

Many important time-critical control systems do not fit the "real-time" stereotype of: being small scale, static, periodic, centralized; performing autonomic monitoring and control of simple devices; and having time frames in the microsecond and millisecond range. Instead, they: are large scale in various dimensions (e.g., millions of lines of source code), dynamic, "mesosynchronous," and distributed; perform closed loop automated control at any level(s) of an enterprise; and typically operate in time frames of hundreds of milliseconds to several minutes.

In mesodynamics [1], the prefix *meso* refers to the middle ground between classical physics and quantum physics. By *mesosynchronous* real-time systems we mean those that are in the middle ground between totally synchronous – in the sense of having only static, periodic, time-driven, activities (e.g., [2]) and totally asynchronous – in the sense of having only dynamic, aperiodic, event-driven, activities. The real world has many important real-time control systems at various points in this mesosynchronous middle ground. The derivation of "mesosynchronous" from "mesodynamics" also reflects that synchronous real-time computing, like classical physics, is relatively well understood, while asynchronous real-time computing, like quantum mechanics, requires a paradigm shift and thus a great deal more research and development on methodologies and formalisms.

It might be tempting to erroneously interpret "mesosynchronous" as meaning that a system is composed of separate traditional synchronous static hard real-time, and asynchronous dynamic, non-real-time parts. While mesosynchronous systems normally do have traditional synchronous hard real-time parts, the asynchronous parts are just as "real-time" as the synchronous ones are. And some parts are neither synchronous nor asynchronous – or are both. Properly speaking, a *real-time* activity is one that has a completion time constraint. Asynchronous activities may have deadlines, even hard deadlines that if missed result in operational failures – assurances about their timeliness are based on adherence to resource management policies, and are almost always unavoidably non-deterministic. More commonly, these activities have softer but more complex time constraints, and sequencing optimality criteria that are softer but more complex than simply always meeting all deadlines (e.g., minimize the expected completion time tardiness accord-

ing to activity importance). That does not mean these activities are in any way less "important" or less mission-critical or even less safety-critical than the synchronous activities – indeed, quite the contrary.

The application domain of primary interest to us is defense. Defense systems and applications have always had by far the most challenging time-critical resource management problems – and are MITRE's predominant focus. Examples include: multi-role and multi-mode sensors (e.g., phased array radars such as MP-RTIP [3]), combat and surveillance platform management (e.g., the new E-10A (MC2A) aircraft [4], UAV's and UCAV's [5], the DD(X) class of destroyers [6], the Army Future Combat System vehicle cluster [7], battle management and mission management, (e.g., time-critical targeting [8]), command and control (e.g., AWACS [9]), and network-centric warfare [10].

Time-critical resource management in network-centric warfare is exemplified by the Affordable Moving Surface Target Engagement (AMSTE) system [11], a notional non-classified depiction of which is in Figure 1. AMSTE includes netted airborne and spaceborne Ground Moving Target Indication (GMTI) sensors for tracking a target on the ground, a missile launching fighter aircraft, and a missile guidance aircraft (which is usually one of the sensor aircraft). The targets are moving or alternately moving and stopping. The AMSTE objectives are: to maintain target track from nomination through engagement, from tactically significant standoff ranges; and to provide precision fire control updates to missiles in flight. These objectives require time-critical orchestration of multiple sensors, data links, and the missile. The control loop has about a one second time constraint for sensor to tracker to missile, and about an eight Hz update rate to the missile (the details of an actual AMSTE system are classified).

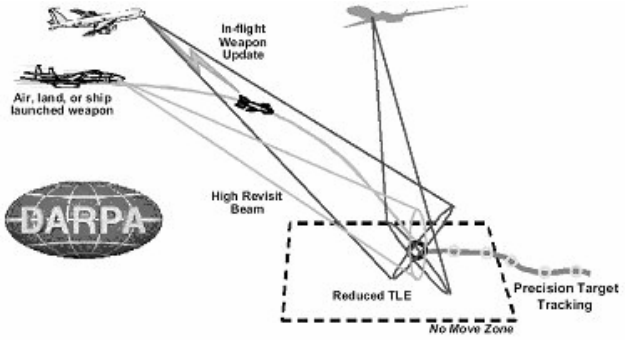


Figure 1. Notional depiction of an AMSTE system

The milliseconds-to-minutes time scale is in a "no man's land" of time-critical resource management theory and practice, between conventional real-time scheduling, and classical logistics (e.g., shop) scheduling with deadlines. Conventional real-time scheduling concepts and techniques: are for static (deterministic), periodic, and (to a degree) sporadic activities, and cannot handle arbitrary asynchrony and dynamic changes in load and resources; time-frames are presumed to be in the microseconds to milliseconds range, which limits resource management algorithm capability. Classical logistics scheduling concepts and techniques [12] include: stochastic as well as deterministic models; but time-frames are in the minutes to hours (usually too computationally intensive for on-line automated resource management); and the use of lateness (deadline - completion time) as the timeliness metric, the linearity of which severely limits time constraint expressiveness.

Time/Utility Functions and Utility Accrual

In 1977 Jensen created a novel scheduling paradigm for dynamic time-critical systems [13] for the U.S. Army Safeguard Command's AN/FPQ-16 Perimeter Acquisition Radar Characterization System [14], located at what is now the USAF Space Command's Cavalier Air Force Station, in Cavalier, North Dakota. The system's radar performance was not up to expectations. Simulated performance with the new scheduling paradigm was sufficiently improved

that the paradigm was deemed to possibly violate SALT II, and hence was not deployed. From then until now, Jensen and his collaborators have continued to develop, refine, and apply the paradigm. At Carnegie Mellon's Computer Science Department, two of Jensen's Ph.D. students wrote their theses on this topic [15,16] – devising new scheduling algorithms, proving theorems about them, simulating them, and implementing them in our Alpha real-time distributed OS kernel [17]. Subsequent enhancements have also been made by other researchers (e.g., [18, 19, 20, 21]).

The two keystone concepts in our paradigm are *time/utility functions* (originally called *time/value* and then *benefit*, functions) and *utility accrual* sequencing (scheduling, dispatching) optimality criteria [13, 22].

A time/utility function (TUF) expresses the utility to the system (derived from AQoS metrics) of completing an activity (e.g., service) as an application- or situation-specific function of when it completes. Deadlines are a simple special case, where full utility is achieved if the activity completes by its deadline, and zero (or some negative) utility is achieved if it completes after its deadline. Four example TUF's are depicted in Figure 2.

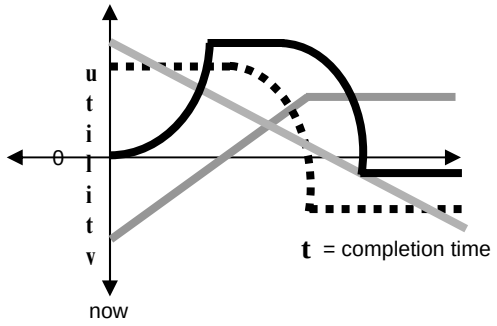


Figure 2. Four example time/utility functions

Utility accrual (UA) based sequencing algorithms sequence activities according to optimality criteria based on accruing utility – such as maximizing the sum of the utilities (plus satisfying dependencies such as precedence, resource constraints, etc.). The algorithms take into account the known or expected durations of the activities, and may be either deterministic or non-deterministic (e.g., stochastic).

Worked Examples

To illustrate the use of this paradigm, we summarize two significant (unclassified) demonstration applications we implemented successfully in collaboration with application domain experts.

AWACS Surveillance Tracker

The first application is an AWACS surveillance tracker [23], implemented jointly by the MITRE Corporation and the Open Group. AWACS is an airborne radar system with varied missions, including air surveillance. Surveillance missions generate aircraft tracks for battle management and command and control. It is very common for there to be too many radar reports for the computing system to process, which causes sectors of the sky to "go blank". Higher performance computing only helps somewhat, because the potential report load and the need for ever-better tracking algorithms will always exceed all the computing capacity. Currently, operators have knowledge-intensive manual work-arounds for certain overload situations.

The surveillance mission includes a number of activities; for brevity here, we consider only the most computationally demanding activity, association, which associates radar reports with tracks. There is a multiplicity of concurrent association threads, one for each current radar report.

There are two sensors (radar and IFF) sweeping 180º out of phase with a 10-second period, which suggests the association TUF has a "critical time" at the 10-second period length, at least two distinct non-zero utilities before the critical time, and a third distinct, lower, utility after the critical time.

Prior to the critical time, processing a sensor report for one of these tracks in under five seconds (half the sweep period) would provide better data for the corresponding report from the out-of-phase sensor. So the utility decreases with time. The TUF had to decrease linearly due to an implementation artifact in this experimental system – the OS (OSF/RI's MK7 [24]) TUF scheduling algorithm allowed only one critical time. The slope, and thus U_2, were derived empirically (systematizing this process is one of our current research topics).

After the critical time, utility is zero, because newer sensor data has probably arrived if the processing load in one sensor sweep period is so heavy that it couldn't be completed, probably the load will be about same in next period, so there will be no capacity to also process data from the previous sweep. A tracker that could process older as well as current data would be significantly more complex and probably delay the track update. The resulting TUF shape is shown in Figure 3.

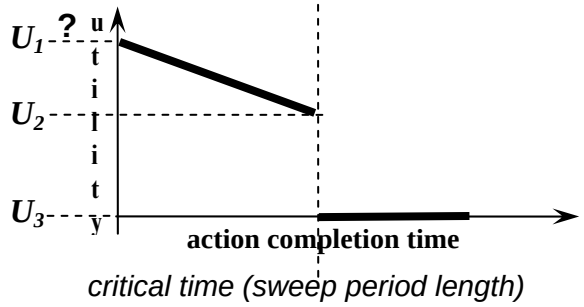


Figure 3. Association time/utility function shape

Next, the utility value U_1 had to be determined. To do that, we used well-established surveillance tracker AQoS metrics, which are based on the tracker domain experts' knowledge about how to manually attempt to resolve contention for resources:

- Don't drop tracks, because they are expensive to re-create
- User-identified "important" tracks receive preference
- User-identified "important" geographic regions receive preference
- Maneuvering tracks need to be updated more frequently than non-maneuvering tracks
- Potentially high threat tracks receive preference
- High speed tracks receive preference
- Tracks with poor state estimates receive preference.

Conventionally, tracker domain experts aren't provided with computation technology that would give them incentives to use their knowledge to understand and express behavioral options and trade-offs in the face of dynamic uncertainties (i.e., gracefully handling overloads) that plague current trackers. Our intention was to provide technology that would utilize tracking domain expertise sufficient to automate the adaptive assignment of the right resources to the right tasks at the right time.

We chose three (to limit the complexity of the experimental system) AQoS metrics for an AWACS surveillance tracking application (a decision with which the domain experts concurred):

- Quality – 0 to 7: a traditional measure of the amount of recent sensor data incorporated in a track record, and incremented or decremented after each radar scan
- Accuracy – "high" or "low": a measure of the uncertainty of the estimate of a track's position and velocity and derived from Kalman filter processing
- Importance – "high" or "low": traditionally, operator-identified based on geography, threat, and other characteristics.

The twelve combinations of these three AQoS metrics are diagrammed in Figure 4.

The initial utility U_1 of an association for a track report is derived from track AQoS metrics by gedanken experiments. The utilities range from a low of 10 to a high of 6000 – meaning that 600 times more utility is gained by performing an association for a low accuracy, high importance, poor quality track, than for a high accuracy, low importance, high quality track.

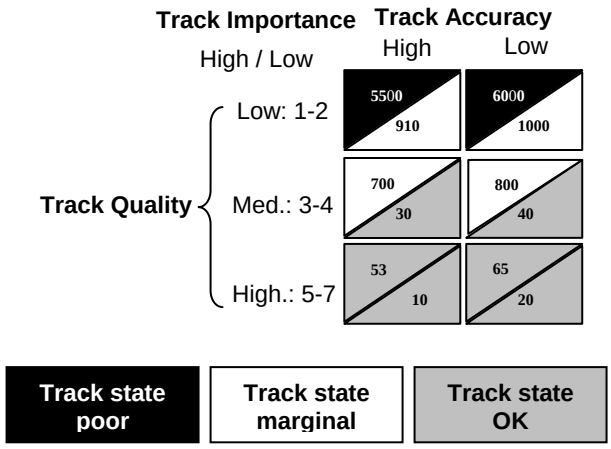


Figure 4. The 12 Combinations of track AQoS's and their relative utilities

All the association (and other) threads are scheduled based on their utility functions. For the association threads, the tracking application selects the established TUF from the scheduler's library of shapes. The tracking application does a look-up in the utility U_1 table for each association thread before calling the scheduler. A utility accrual based processor scheduling discipline (in the OS, in this system) schedules threads according to a heuristic that attempts to maximize total accrued (in this case, summed) utility.

This surveillance tracking application was implemented on a research OS (the Open Software Foundation's MK7) having a scheduling framework that allowed different disciplines to be used. Figure 5 shows two critical tracker AQoS metrics, the number of dropped tracks and track quality, for more and less important threads, using three scheduling disciplines: FIFO (the discipline used in the currently deployed AWACS tracker); priority (the predominant discipline in real-time computing systems); and one of our utility accrual disciplines.

In the measured cases shown, under non-overload conditions, the tracker handled all tracks as expected and delivered high AQoS in both metrics for all tracks. In an overload scenario, such as when the tracker can only process about 33% of the input, the system delivered essentially perfect track quality for the more important tracks, while delivering a reasonable track quality for the less important tracks. (In some respects, this overload level can be likened to a system where the probability of detecting an airborne object is about 33%.) When the tracker was further constrained so that it could only process about 10% of the input, the system finally dropped some tracks – from the less important track class. No important tracks have been dropped during our demonstrations. In addition, the demonstrations have shown that the tracker also adapts when new resources are added. In that case, which we demonstrate by loosening the time constraints on association processing, the prototype automatically delivers approximately the maximum achievable AQoS.

Note that these results are not easily obtainable with static priority tracking systems. In priority-based trackers, track priorities might reasonably be set according to track importance, where high importance implies high priority. In our tracker, scheduling decisions are based on both importance and timeliness, and even relatively unimportant tracks can have very high application utility in a surveillance mission – an outcome that would not be possible with straightforward priority-based scheduling.

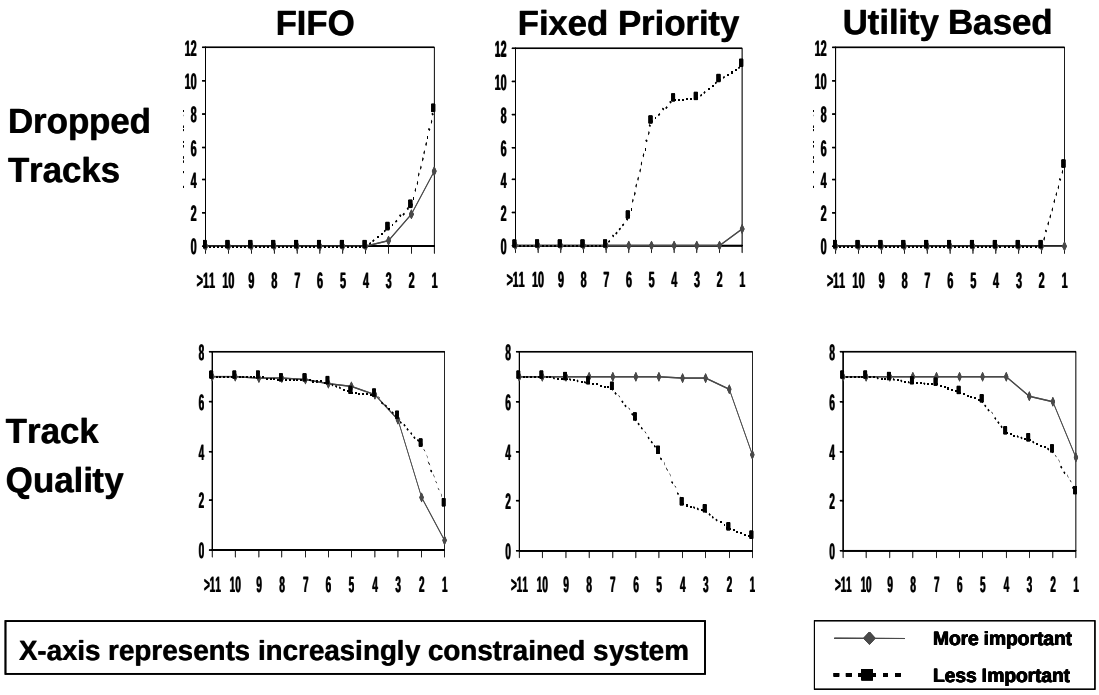


Figure 5. Comparison of AWACS surveillance tracker AQoS metrics for different scheduling disciplines

Battle Management for Coastal Air Defense

The second application is a notional battle management system for coastal defense from cruise missiles and bombers [25]. It was implemented collaboratively by the General Dynamics Corporation and Jensen's Archons Project at Carnegie Mellon University's Computer Science Department. It is included here to further illustrate the dynamic adaptivity possible with time/utility functions, which would be difficult to achieve with priorities or even deadlines.

This system's mission is to destroy incoming hostile cruise missiles (for brevity here we will disregard the hostile bombers), by using guided interceptor missiles. There may be more cruise missiles and decoys than can be intercepted, due to the number of cruise missiles, insufficient interceptors, and insufficient computational resources for battle management. Cruise missiles maneuver during flight, but do not try to evade the interceptors. Interceptors are guided by airborne defenders using airborne (AWACS, etc.), spaceborne, and ground based, sensor platform data.

The cruise missile defense (CMD) application quality of service (AQoS) metric here is the weapon spherical error probable (WSEP). Spherical error probable (SEP) is the radius in meters of a sphere centered on a point (e.g., the cruise missile) within which the true value of an estimated point (e.g., the interceptor missile) will lie with a probability of 0.5. The WSEP is an SEP radius around the cruise missile chosen such that if the interceptor gets within that distance, the interceptor's terminal guidance system can take over and with ρ=0.5 cause the interceptor to either impact the cruise missile or detonate close enough to the cruise missile to destroy it.

Some CMD AQoS optimality objectives are:

- minimizing WSEP
- intercepting sooner (further from the cruise missile's target) is better (EMP, etc.)

- intercepting before the cruise missile reaches its target is better
- intercepting away from an adverse geographical location (e.g., over a populated area) is better
- intercepting before the blast damage to the cruise missile's target from the combined cruise missile and interceptor missile would exceed the blast damage from the cruise missile explosion alone is better.

WSEP is affected by several factors, including:

- guidance updates to the interceptor are repetitive but not necessarily periodic or even sporadic, and need to occur more often as the interceptor gets closer to the cruise missile
- it becomes more important to use the most recent information about the positions and velocities of the missiles as the distance between the cruise missile and the interceptor decreases
- some cruise missile targets, and thus cruise missiles, are more important than others.

The CMD application consists of a number of activities, as depicted in Figure 6.

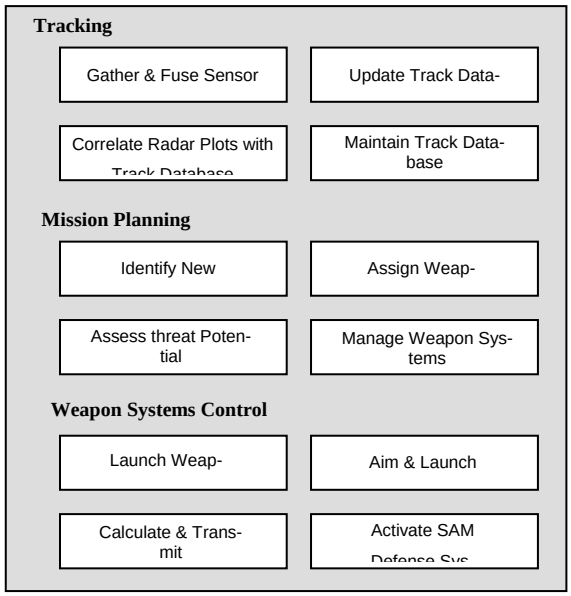
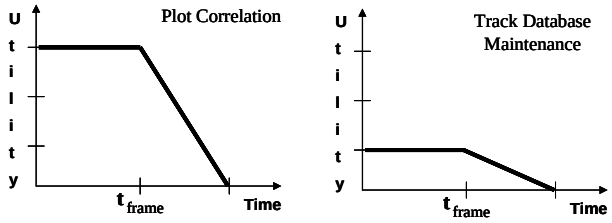


Figure 6. The coastal air defense battle management activities

Some activities resemble corresponding activities in the AWACS surveillance tracker. The plot correlation and track database maintenance threads have critical times corresponding to the radar frame arrival rate. In both cases, it is better if the processing is completed before the next frame of sensor data arrives. It is acceptable for the processing to slip as much as one additional time frame under extreme overload situations. The plot correlation activity has a much greater utility to the system under overload conditions. The TUF's for those two threads are shown in Figures 7a and 7b.



Figures 7 a,b. The plot correlation and track database time/utility functions

But the timeliness requirements for the interceptor missile control threads are more complex because they vary over the course of an interception engagement. After an interceptor is launched, the guidance control threads must issue timely aperiodic course updates to ensure a successful intercept. The required timeliness of these updates, and the importance of completing the course corrections at the desired time, change as the distance decreases between the interceptor and the cruise missile, and between the cruise missile and the coastline. Each interceptor's control thread changes three parameters of its shape during the engagement, as shown in Figure 8. Figure 9 shows three representative snapshots of this adaptivity as it progresses from right (launch) to left (intercept).

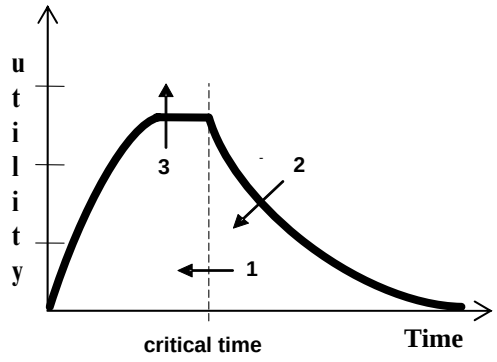


Figure 8. The interceptor time/utility function has three shape parameters that adapt during the engagement

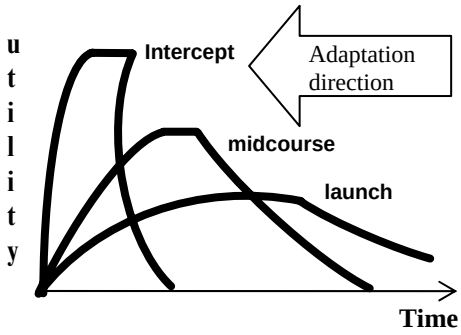


Figure 9. Three representative snapshots of the interceptor time/utility function adapting during the engagement

This effect is extremely difficult to achieve by manipulating priorities or deadlines.

Conclusion

Our current research objective on this topic is to advance this paradigm's concepts and techniques, and to explore its applications in the BM/C^2 domain. We are devising new resource management algorithms, proving timeliness and other properties, doing simulations of them, building and measuring experimental implementations of them, devising a methodology for users to apply the paradigm, and constructing a proof of concept software tool to facilitate the methodology.

References

[1] R.D. Peters, http://mathforum.org/library/view/16558.html

[2] H. Kopetz, *Real-Time Systems, Design Principles for Distributed Embedded Applications*, Chapter 8, "The Time-Triggered Protocols," Kluwer Academic Publishers, 1997.

[3] http://www.globalsecurity.org/intell/systems/mp-rtip.htm

[4] http://esc.hanscom.af.mil/esc-mc2a/

[5] http://www.globalaircraft.org/planes/?type=uav

[6] http://peoships.crane.navy.mil/ddx/

[7] http://www.darpa.mil/fcs/

[8] http://www.globalsecurity.org/space/systems/bmc3i.htm

[9] http://www.globalsecurity.org/military/systems/aircraft/e-3.htm

[10] http://www.dodccrp.org/ncwPages/ncwPage.html

[11] http://www.globalsecurity.org/military/systems/munitions/amste.htm

[12] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 2nd edition, Prentice Hall, 2002, ISBN 0-130-28138-7.

[13] M. G. Gouda, Y-W. Han, E. D. Jensen, W. D. Johnson, and R.Y. Kain, *Distributed Data Processing Technology, Vol. IV, Applications of DDP Technology to BMD: Architectures and Algorithms*, Honeywell Systems and Research Center, Minneapolis, MN. September 1977.

[14] http://srmsc.org/

[15] C. D. Locke, *Best-Effort Decision Making for Real-Time Scheduling*,, Ph.D. Thesis, CMUCS-86-134, Department of Computer Science, Carnegie Mellon University, 1986.

[16] R. K. Clark, *Scheduling Dependent Real-Time Activites,* Ph.D. Thesis, CMUCS-90-155, School of Computer Science, Carnegie Mellon University, 1990.

[17] R. K. Clark, E. D. Jensen and F. D. Reynolds, “An Architectural Overview of the Alpha Real-Time Distributed Kernel”, *USENIX Workshop on Microkernels and other Kernel Architectures*, pp 200-208, 1993.

[18] P. Li, B. Ravindran, H. Wu, and E. D. Jensen, A utility accrual scheduling algorithm for real-time activities with mutual exclusion resource constraints, *IEEE Transactions on Computers*. Submitted August 2003.

[19] D. Mosse, M. E. Pollack, and Y. Ronen, “Value-density algorithm to handle transient overloads in scheduling," in *Proc. Euromicro Conference on Real-Time Systems*, June 1999, pp. 278-286.

[20] W. T. Strayer, “Function-driven scheduling: A general framework for expressing and analysis of scheduling," Ph.D. dissertation, University of Virginia, May 1992, department of Computer Science.

[21] S. A. Aldarmi and A. Burns, “Dynamic value-density for scheduling real-time systems," in *Proc. of Euromicro Conference on Real-Time Systems*, June 1999, pp. 270-277.

[22] E. D. Jensen, C. D. Locke, and H. Tokuda, “A Time-Driven Scheduling Model for Real-Time Systems,” *Proc. Real-Time Systems Conference*, IEEE, December 1985.

[23] R. Clark, E. D. Jensen, A. Kanevsky, J. Maurer, P. Wallace, T. Wheeler, Y. Zhang, D. Wells, T. Lawrence, and P. Hurley, “An adaptive, distributed airborne tracking system," in *Proceedings of IEEE International Workshop on Parallel and Distributed Real-Time Systems*, ser. Lecture Notes in Computer Science, vol. 1586. Springer-Verlag, April 1999, pp. 353-362.

[24] D. Wells, “A Trusted, Scalable, Real-Time Operating System,” *Dual-Use Technologies and Applications Conference Proceedings*, pp II 262-270, Utica, NY, 1994.

[25] D. P. Maynard, S. E. Shipman, and R. K. Clark. et al., “An example real-time command, control, and battle management application for alpha,” Technical Report Archons Project TR-88121, CMU Computer Science Department, December 1988.

MITRE Technology Program

Information Management

Application QoS Based Time-Critical Automated Resource Management in BM/C^2 Systems

E. Douglas Jensen

The MITRE Corporation

jensen@[mitre,real-time].org
http://www.[real-time, mitre].org

Revised 9 June 04

MITRE

Summary

- This presents a novel paradigm for expressing, enforcing, and formally reasoning about time-criticality of machine-to-machine resource management in battle management (BM) and C^2 systems
- Such systems are largely dynamic and asynchronous, and have time-critical actions in the $O(10^{-1} – 10^3)$ seconds
- Thus they fall into a neglected gap between traditional static periodic "real-time" systems, and traditional "any time" scheduling/planning systems (e.g., for logistics)
- The paradigm uses *application*-level QoS (AQoS) metrics (such as track quality, circular error probable, etc.) to derive utility functions for completing tasks,

  and then uses those utility functions for resource management
- This paradigm has been successfully employed in several experimental BM/C^2 demonstration systems
- It is the topic of active research in academia and industry

Many important time-critical systems (such as for BM/C^2) have significant dynamic actions

- Many important time-critical control systems do not fit the "real-time" stereotype of
  - small scale
  - static
  - periodic
  - centralized
  - performing monitoring and control of simple devices
  - time frames in the microsecond and millisecond range
- Instead, they are
  - large scale in various dimensions
  - dynamic
  - "mesosynchronous"
  - distributed
  - performing closed loop machine-to-machine control at any level(s) of an enterprise
  - operating in the second to minutes time frame

Priorities have severe limitations, especially in dynamic systems

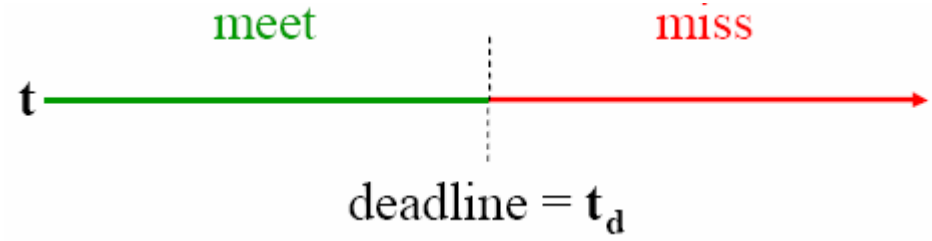
- Priorities are widely used in time-critical systems, but they have major disadvantages, including
  - priority assignments are not modular – they require global knowledge of all other priority assignments (whereas time constraints, such as deadlines, do not)
  - the granularity of time constraints is typically much finer than that of priority ranges, and mapping time constraints to priorities is NP-hard
  - semantics are associated with priorities by the users, the system and application software, and the hardware
    - sometimes priorities (artificially) denote urgency
    - sometimes priorities denote relative importance
    - sometimes priorities denote execution precedence
- Managing the assignments and changing of priorities is one of the most notoriously difficult and time-consuming activities in the life cycle of systems

Priorities are a Procrustean Bed

- Priorities are the primary mechanism offered in COTS (and application) software for managing timeliness, so designers and users must try to force-fit time constraints into priorities

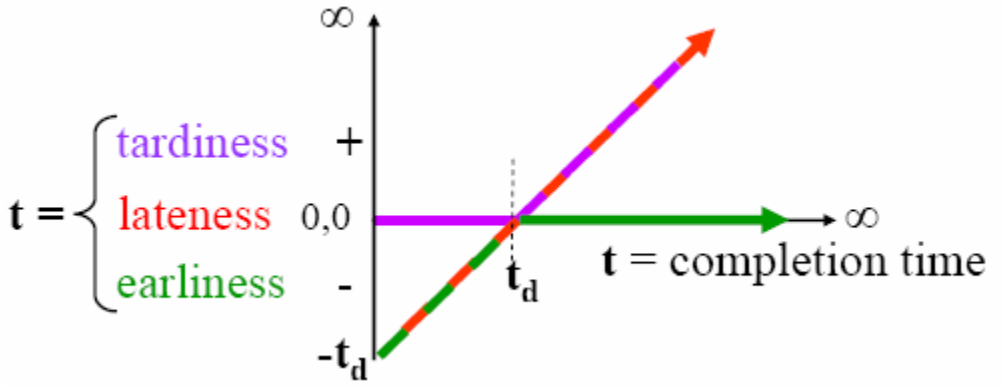
Deadlines are an explicit time constraint but have limited expressiveness

- Deadlines, as popularly spoken of in "hard" real-time computing, are only binary: an action either meets or misses it



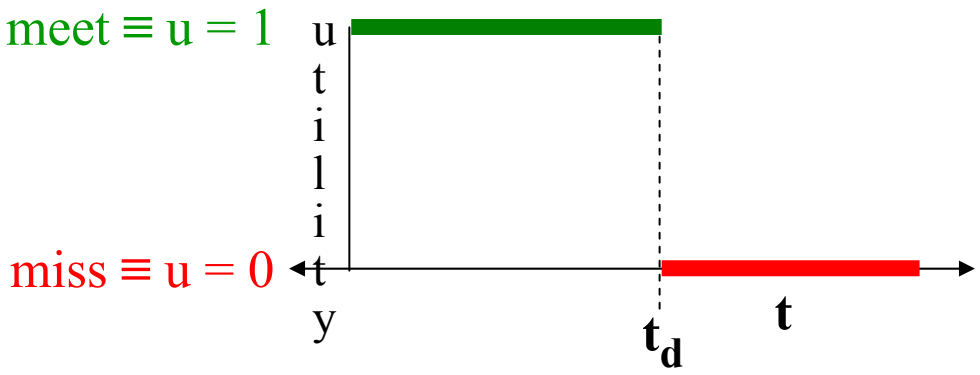
but in most real world cases, *lateness* is the actual criterion

- Scheduling theory deals with lateness, but deadlines have only
  - linear timeliness metric, *lateness* = completion time - deadline
  - single inflection point metric, *tardiness* = max[0,lateness]

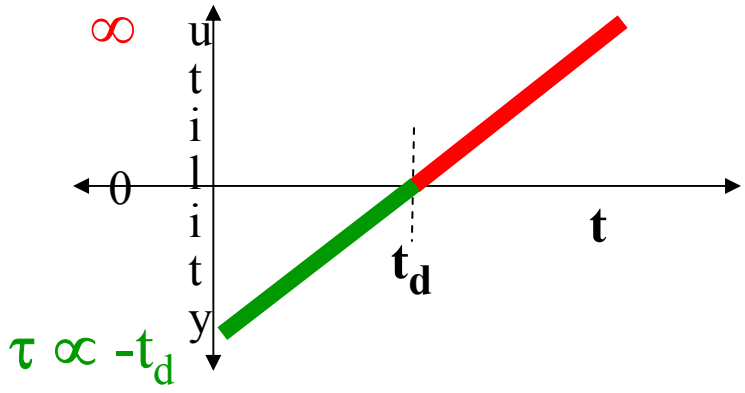


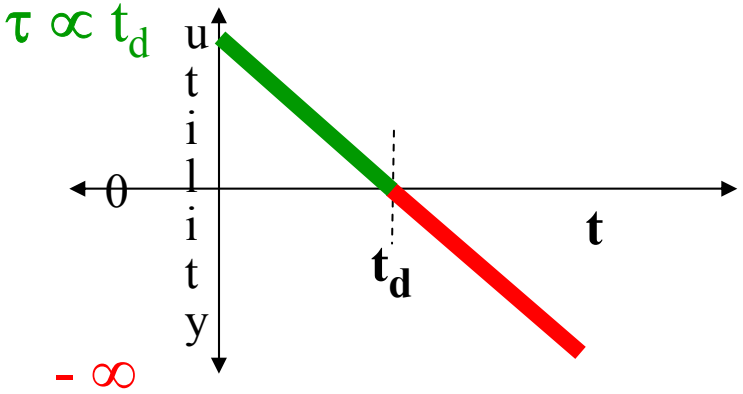
"Hard" deadlines and general deadlines can be represented by utility as a function of time

- A "hard" deadline is a binary unit-valued downward step



- A general deadline in terms of lateness and negative lateness

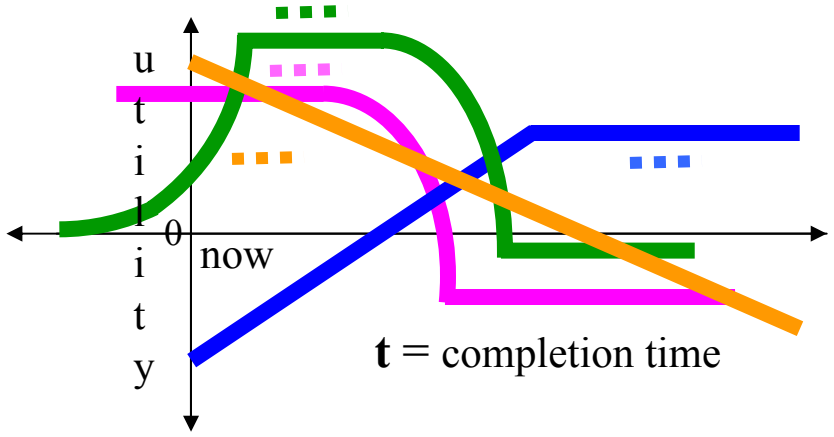




The two keystone concepts in our paradigm are time/utility functions and utility accrual scheduling

- **Time/utility functions (TUF's)**
  - **express the utility to the system (derived from AQoS metrics) of completing an activity (e.g., service) as an application- or situation-specific function of when it completes**

Example



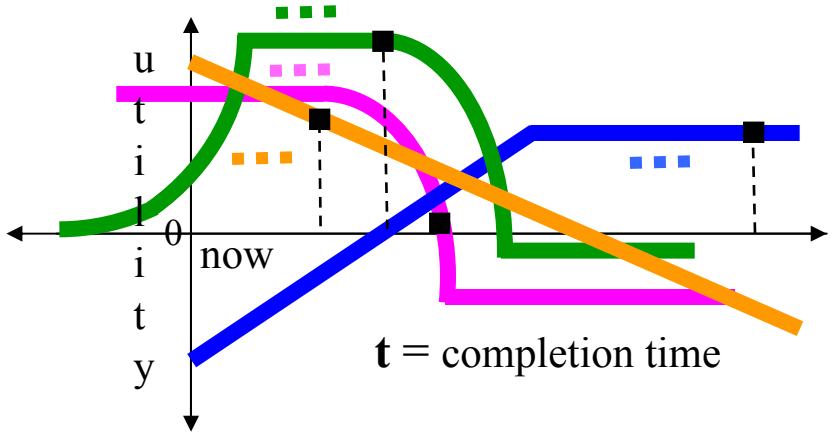
General time/utility functions

▪▪▪ Expected or max execution time

The two keystone concepts in our paradigm are time/utility functions and utility accrual scheduling

- Time/utility functions (TUF's)
  - express the utility to the system (derived from AQoS) of completing an activity (e.g., service) as an application- or situation-specific function of when it completes
- **Utility accrual (UA) scheduling algorithms**
  - **schedule activities according to optimality criteria based on**
    - **accruing utility – such as maximizing the sum of the utilities**
    - **satisfying dependencies such as resource constraints, etc.**

Example



General time/utility functions

▪▪▪ Expected or max execution time
■ Example scheduled completion times

Schedule to maximize
$U = \sum u_i$ ■

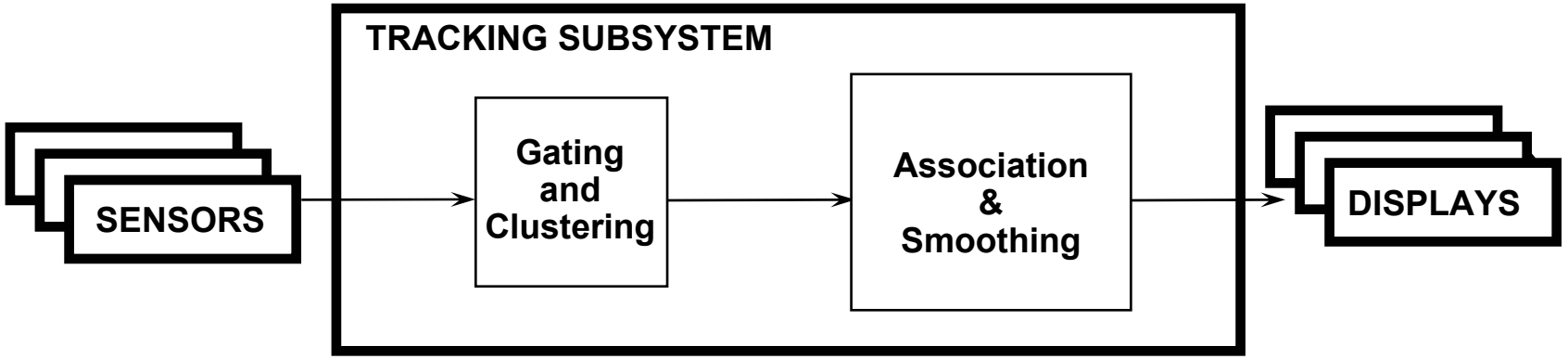
Worked examples using TUF/UA scheduling

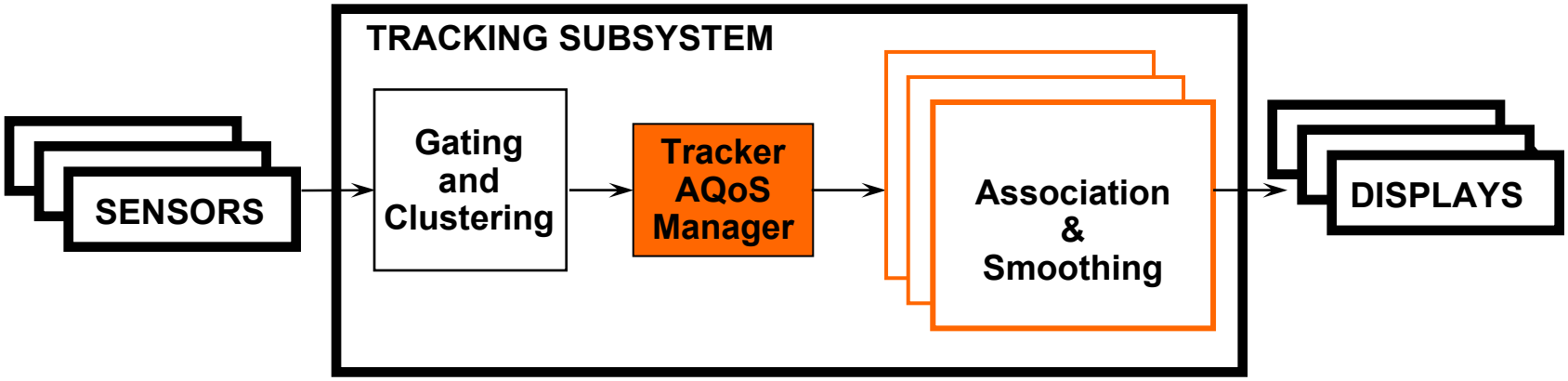
- This paradigm has been applied in significant size BM/C^2 demonstration systems
- Next we illustrate the paradigm in the context of an air surveillance tracking application
- Then we show one facet of the paradigm's use – not employed in the surveillance tracker – in a cruise missile defense application
- Another major demonstration application is currently being constructed but cannot be discussed here

TUF/UA sequencing paradigm worked example 1: AWACS air surveillance mode tracker

- MITRE (with collaboration from the Open Group) applied our paradigm in a demonstration AWACS system
- Implemented the AWACS air surveillance mission
- It is easy and common for there to be so many sensor reports that the system becomes computationally overloaded, which causes sectors of the sky to "go blank"
- Currently, operators have knowledge-intensive manual work-arounds for certain overload situations
- Our objective was to improve graceful overload handling by automatically
  - applying the right computational resources
  - to the right tracks
  - at the right times

Tracking system: original and adaptive

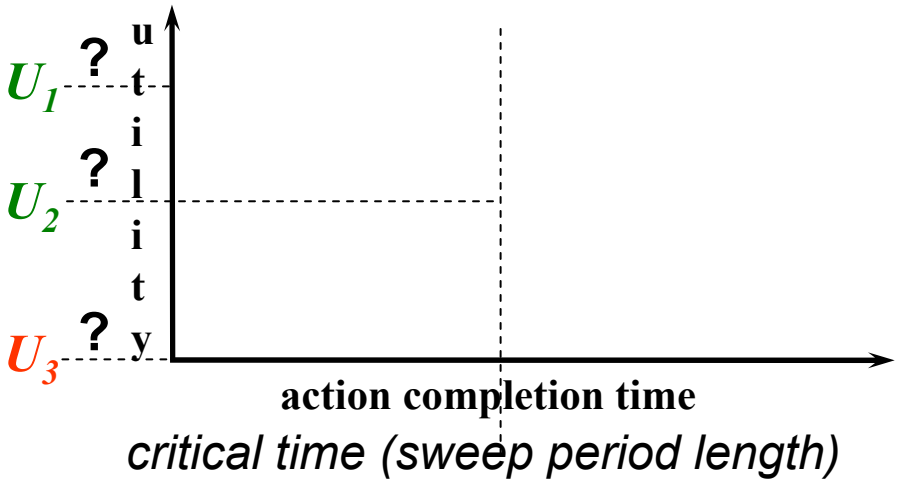




The AWACS sensor properties imply a general utility function for the association computation

- Association is the most computationally demanding part of tracking, so we focus on that in this presentation
- There are two sensors (radar and IFF) sweeping 180° out of phase with a 10-second period,

  which suggests the TUF has
  - a "critical time" at the 10-second period length
  - at least two distinct non-zero utilities before the critical time
  - a third distinct, lower, utility after the critical time



Determine the thread TUF shape prior to the critical time

- Prior to the critical time
  - processing a sensor report for one of these tracks in under five seconds (half the sweep period) would provide better data for the corresponding report from the out-of-phase sensor

    so the utility decreases with time
  - the TUF had to decrease linearly due to an implementation artifact in this experimental system –

    the OS (OSF/RI's MK7.3A) TUF scheduling algorithm allowed only one critical time
  - the slope was derived empirically

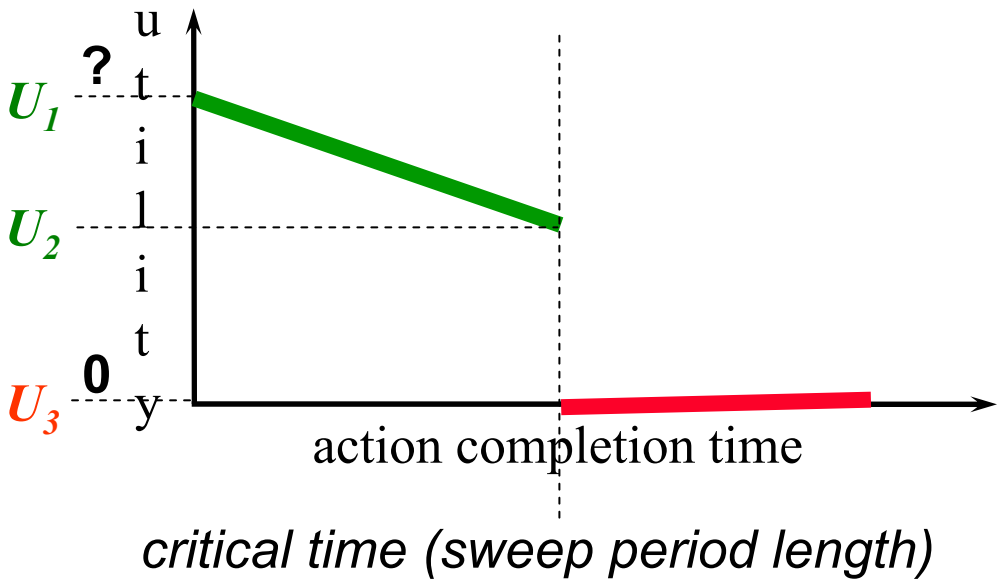
Determine the thread TUF shape after the critical time

- After the critical time
  - utility is zero, because newer sensor data has probably arrived
  - if the processing load in one sensor sweep period is so heavy that it couldn't be completed,

    probably the load will be about same in next period, so there will be no capacity to also process data from the previous sweep
  - a tracker that could process older as well as current data would
    - be significantly more complex
    - probably delay the track update

That established the TUF shape for the tracker's association threads

- A critical time at the sweep period length
- Linearly decreasing utility until the critical time
- Zero utility after the critical time
- Next, the utility value U_1 had to be determined



Tracker domain experts' preferences in terms of track QoS metrics imply the thread utility values

- Don't drop tracks, because they are expensive to re-create
- User-identified "important" tracks receive preference
- User-identified "important" geographic regions receive preference
- Maneuvering tracks need to be updated more frequently than non-maneuvering tracks
- Potentially high threat tracks receive preference
- High speed tracks receive preference
- Tracks with poor state estimates receive preference

Three application-level QoS metrics for an AWACS surveillance tracking application were chosen

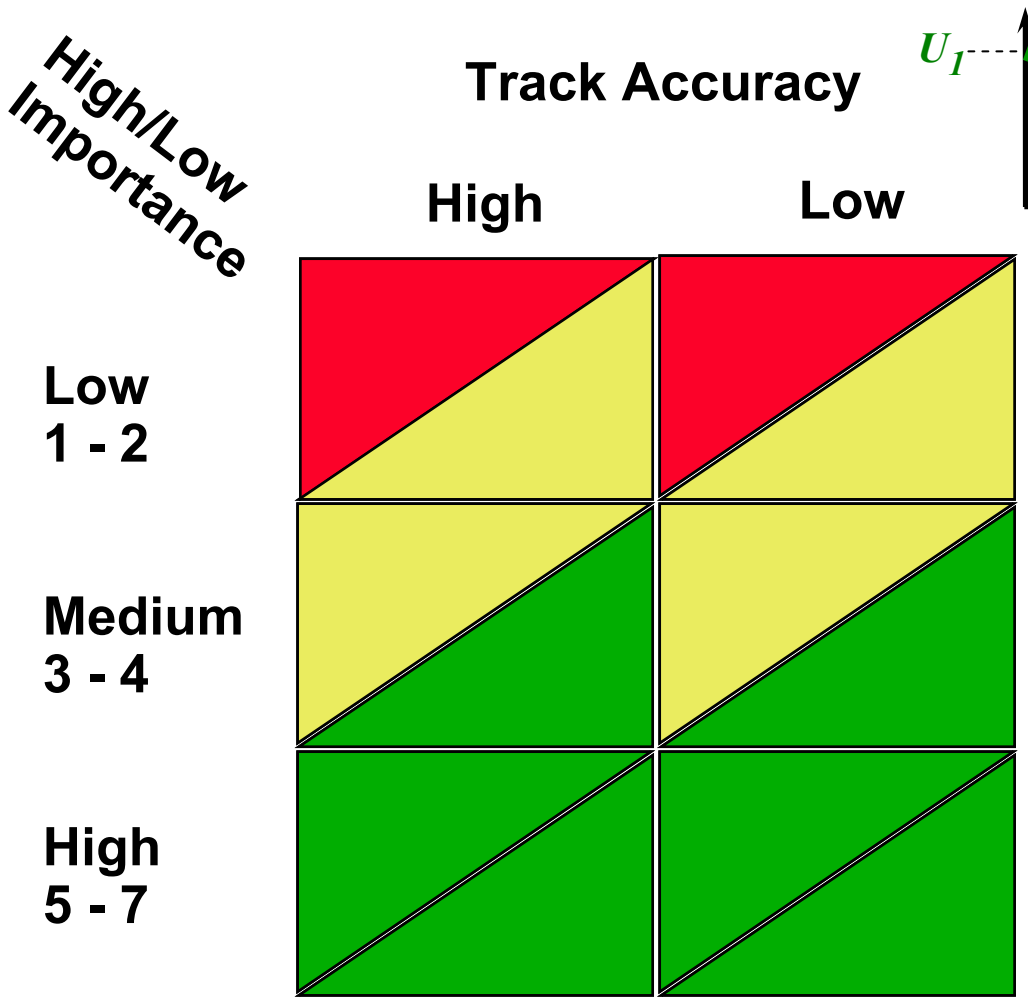
- Quality – 0 to 7
  - traditional measure of the amount of recent sensor data incorporated in a track record
  - incremented or decremented after each radar scan
- Accuracy – "high" or "low"
  - a measure of the uncertainty of the estimate of a track's position and velocity
  - derived from traditional Kalman filter processing
- Importance – "high" or "low"
  - traditionally, operator-identified based on geography, threat, and other characteristics

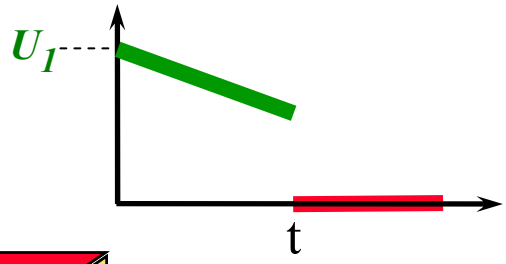
We established 12 combinations of track AQoS metrics

Track state poor

Track state marginal

Track state OK





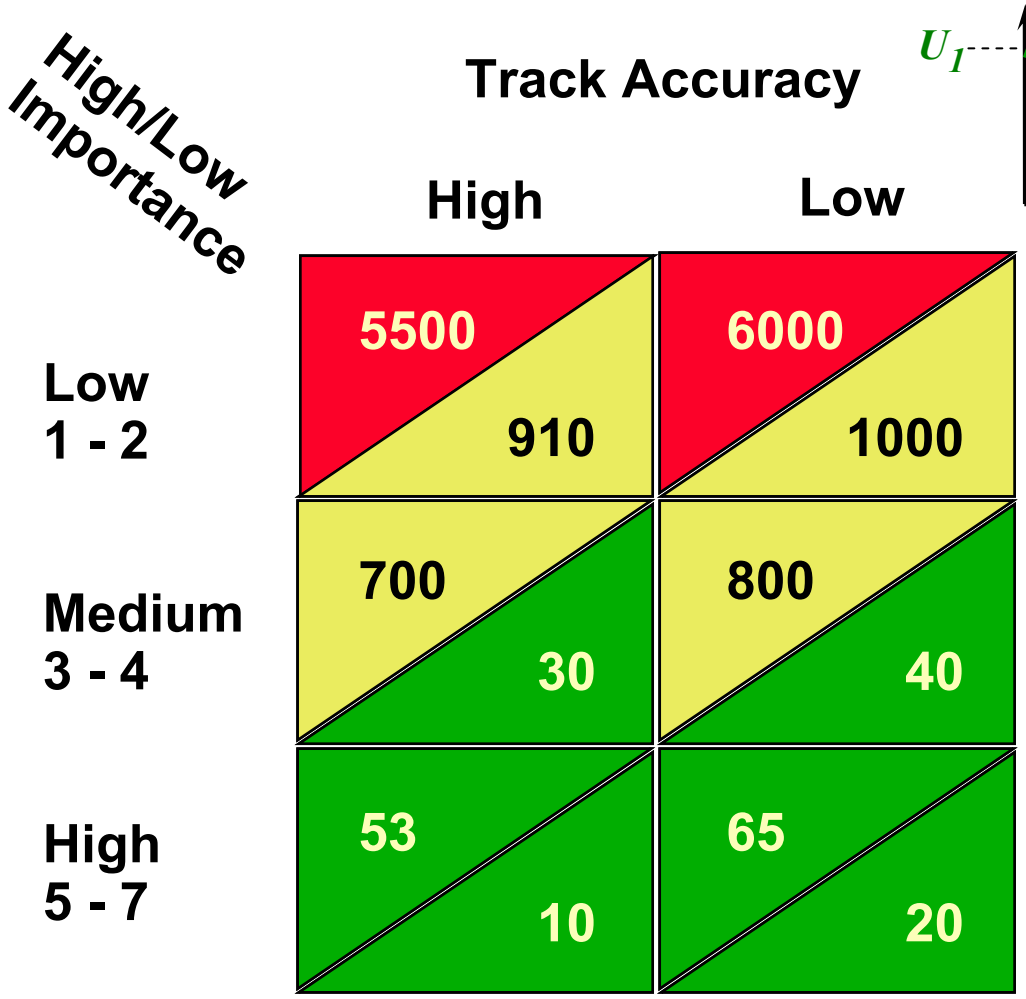
What are the relative utilities of these 12 cases of tracks?

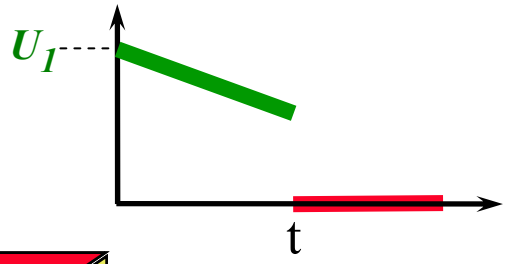
The initial utility U_1 of an association for a track report is derived from track AQoS metrics by gedanken experiments

Track state poor

Track state marginal

Track state OK





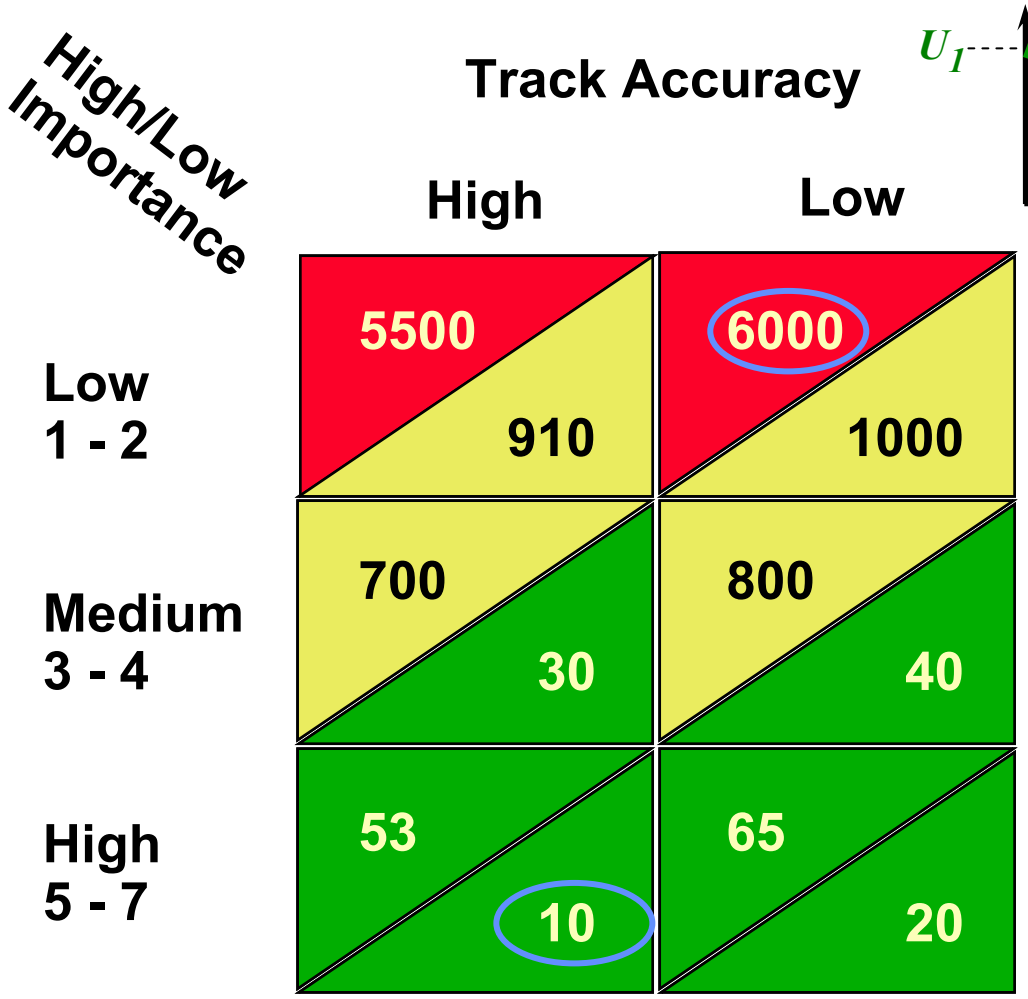
Domain experts judgment on the relative utilities of these 12 cases of tracks

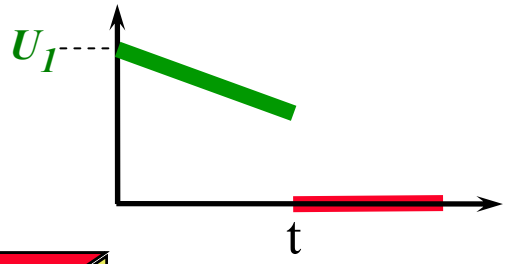
The initial utility U_1 of an association for a track report is derived from track AQoS metrics by gedanken experiments

Track state poor

Track state marginal

Track state OK





E.g., completing an association for a high importance, low accuracy, low quality track yields 600 times more utility than for a low importance, high quality, high accuracy track

Domain experts judgment on the relative utilities of these 12 cases of tracks

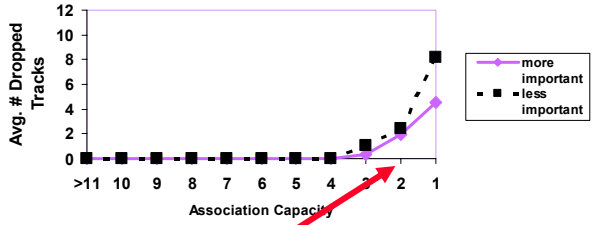
The association (and other) threads are scheduled based on their utility functions

- For the association threads, the tracking application selects the established TUF from the OS scheduler's library of shapes
- The tracking application does a look-up in the utility U_1 table for each association thread before calling the OS scheduler
- A utility-based processor-scheduling policy in the OS schedules threads according to a heuristic that attempts to maximize total accrued (in this case, summed) utility

Utility-based scheduling provided better AQoS than traditional FIFO and priority scheduling

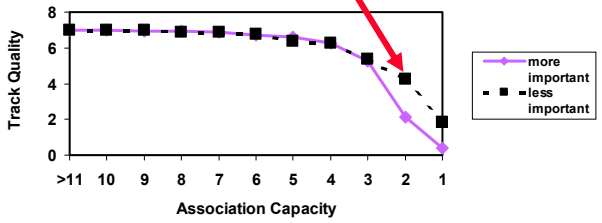
FIFO

Avg. # Dropped Tracks versus Association Capacity For FIFO Priority



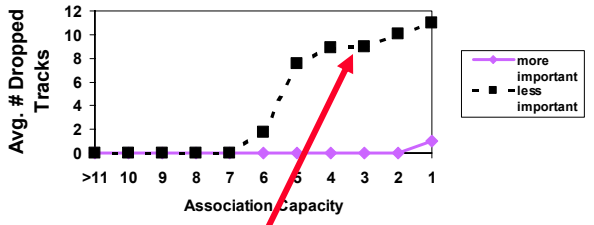
High Priority Tracks Were Dropped and have Bad TQ

Track Quality versus Association Capacity For FIFO Priority



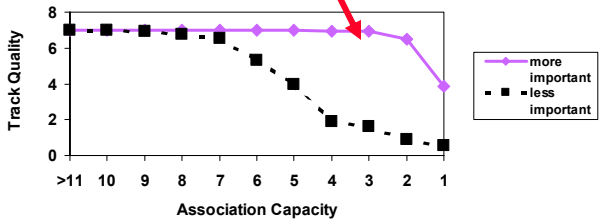
Priority

Avg. # Dropped Tracks versus Association Capacity For Fixed Priority



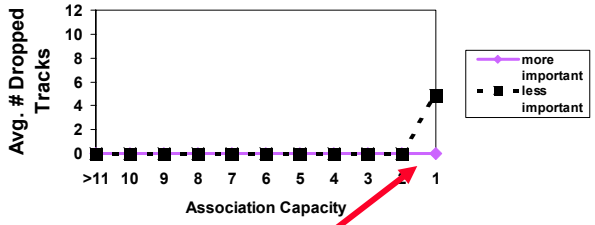
Drop *Low Priority* Tracks to Get Better TQ on *High Priority* Tracks

Track Quality versus Association Capacity For Fixed Priority



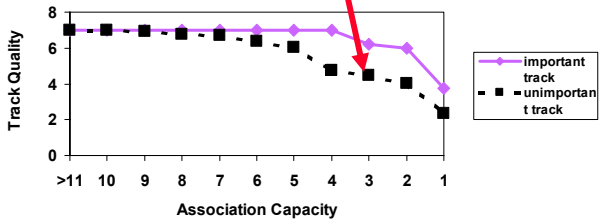
Utility-Based

Avg. # Dropped Tracks versus Association Capacity For Dynamic Priority



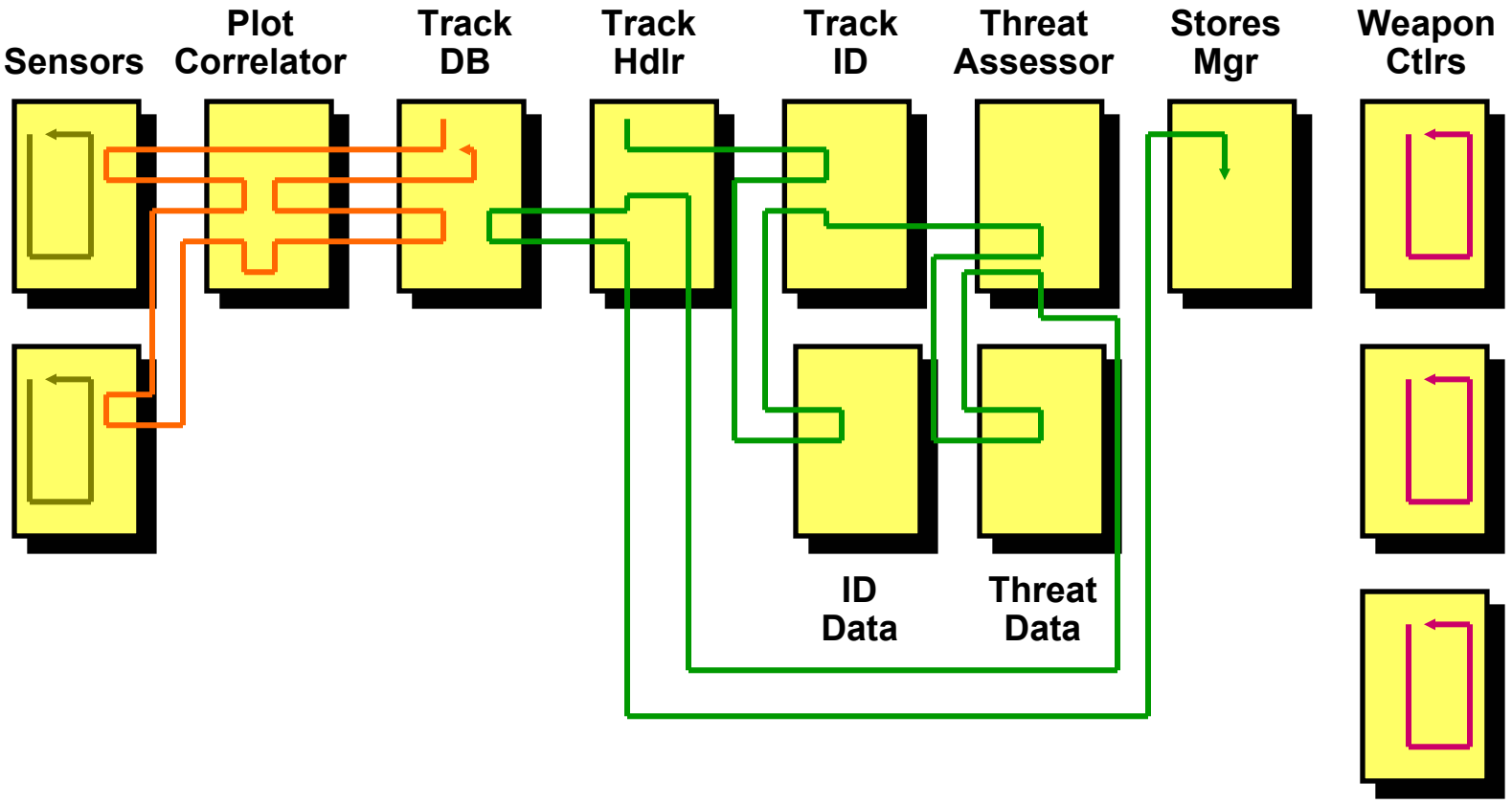
No *High Priority* Tracks Dropped. Overall Better TQ

Track Quality versus Association Capacity For Dynamic Priority



Key:
Track Quality: 0-7 (7 = Ideal)
Association Capacity = # Tracks Processed under Constraint

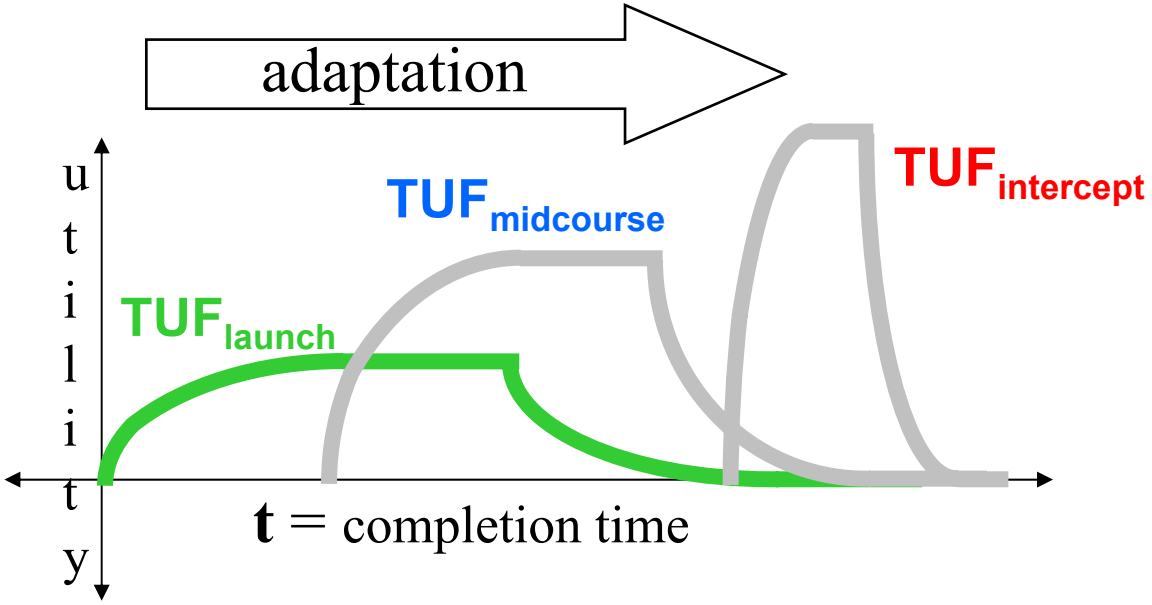
TUF/UA sequencing paradigm worked example 2: cruise missile defense with guided interceptors



Distributable Threads: a programming model for end-to-end timeliness in distributed systems – created by Jensen's CMU Alpha OS team and now in Real-Time CORBA 1.2 (née 2.0)

The timeliness requirements for the interceptor missile control threads vary over the course of an engagement

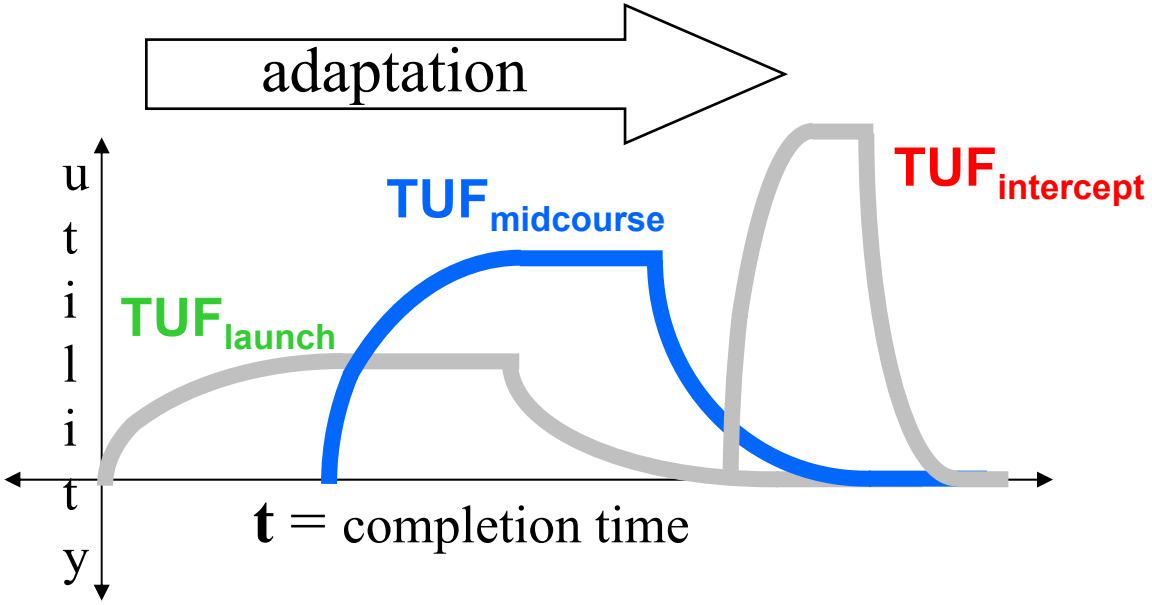
- After launch of the interceptor, the guidance control threads must issue timely repetitive course updates to ensure a successful intercept
- The required timeliness of these updates, and the importance of completing the course corrections at the desired time, change as the distance decreases between the interceptor and the cruise missile, and between the cruise missile and its expected target



- This effect is very difficult to achieve by manipulating priorities

The timeliness requirements for the interceptor missile control threads vary over the course of an engagement

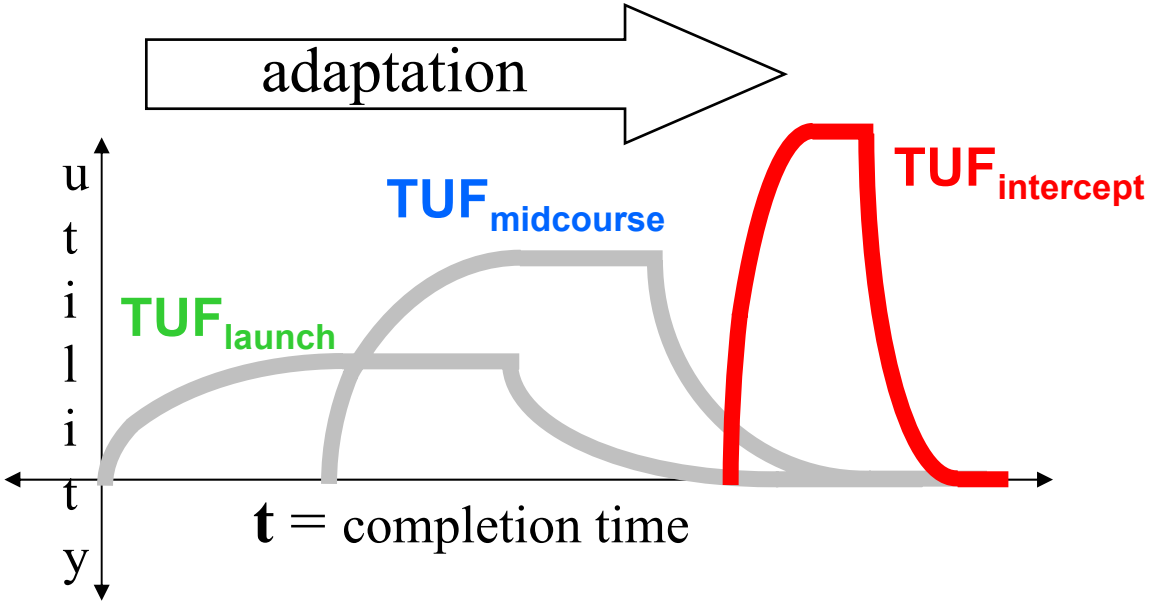
- After launch of the interceptor, the guidance control threads must issue timely repetitive course updates to ensure a successful intercept
- The required timeliness of these updates, and the importance of completing the course corrections at the desired time, change as the distance decreases between the interceptor and the cruise missile, and between the cruise missile and its expected target



- This effect is very difficult to achieve by manipulating priorities

The timeliness requirements for the interceptor missile control threads vary over the course of an engagement

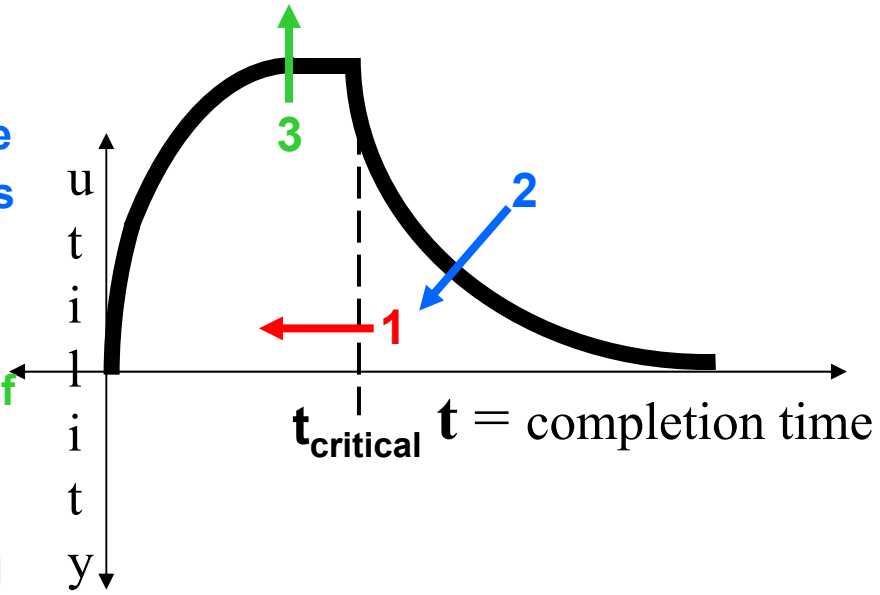
- After launch of the interceptor, the guidance control threads must issue timely repetitive course updates to ensure a successful intercept
- The required timeliness of these updates, and the importance of completing the course corrections at the desired time, change as the distance decreases between the interceptor and the cruise missile, and between the cruise missile and its expected target



- This effect is very difficult to achieve by manipulating priorities

The TUF's for the missile and interceptor control updates change dynamically during the mission

- 1. Variable critical (best) times – course corrections are needed more often as the distance between target and interceptor decreases
- 2. Variable "hardness" – it becomes more important to use the most recent position information as the distance between target and interceptor decreases
- 3. Dynamic maximum – the utility of successfully completing an intercept corresponds to the perceived threat of the target being intercepted



They also have variable importance depending on the threat potential of the target, independent of their timeliness

TUF/UA scheduling can be very cost/effective in adaptively achieving superior AQoS

- TUF time constraints have been shown to be very natural, expressive, and powerful for the designers and programmers of the BM/C^2 applications we have experimented with
- But this paradigm does impose costs
  - TUF's are more complex than priorities
  - UA scheduling is more complex than priority dispatching
- Various application-specific engineering techniques can be used to trade off costs vs. effectiveness

A proof of concept software tool is being produced along with methodology and formalism

- MITRE and our academic collaborator Virginia Tech are developing a proof of concept software tool for
  - creating and manipulating TUF's
  - plugging in various application-specific UA (and other) scheduling algorithms
  - simulating and analyzing the resulting schedules
- One version of this tool is being done in the context of an extant COTS real-time timing analysis product –

  the vendor is interested in the commercialization of our work
- We are also creating, simulating, implementing, measuring, and proving properties of, new UA algorithms –

  published and pre-published papers are available

Conclusion

- Application designers often think in terms of what we refer to as AQoS metrics, but not in a general and methodological way
- Instead, they consider certain metrics and use their domain expertise to attempt to aggregate these into the proper "tuning" of the system
- Thus, they've had few incentives to use their knowledge to understand and express behavioral options in the face of dynamic uncertainties (i.e., gracefully handling overloads) to facilitate automated resource management
- Time/utility functions are more natural, expressive, and realistic for dynamic systems, than priorities and deadlines
- AQoS metrics can be used to derive TUF's
- UA scheduling optimality criteria are powerful and adaptive
- TUF/UA based resource management has been shown to be very promising for dynamic systems such as BM/C^2